

Assignment Report Requirements

- Implement Training of the HMM¹ with smoothing
- Implement Computation of most probable transition and emission of sequence of tags for untagged sentences from trained HMM
- Implement code for evaluation of the HMM
- Implement code that trains, and evaluates a model for 5 corpora
- Added ReadMe file

Design

- 'Main.py' class
- In this class, sentences are imported from the corpora, loaded and stored. Four total distinct corpora are used namely: Brown, Conll-2000, Alpino and Floresta.
- The sentences from the corpora chosen are then split into training and test sets.
- The words and tags for each of the corpora chosen are then extracted and separated into dictionaries using the 'extractWords_and_Tags' method which parses each of the corpus.
- The HMM models for each corpus is created and trained.
- The models created from above are then evaluated on their transition and emission probabilities and the results printed out. Note: Training more models takes quite a long time hence why the number of corpora used is 4.

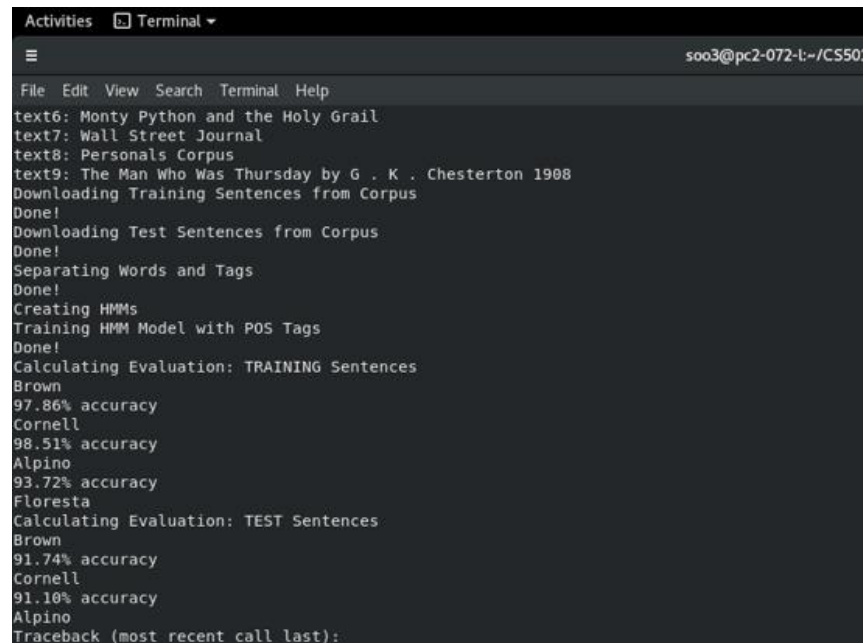
Training

- 'HMM.py' class
- The python program has three classes i.e. HMM.py, main.py and StateClass.py
- In HMM.py, the class is initialised in '__init__' where the set of tags is passed through the method. The method adds the 'start' and 'end' characters to help indicate the start and end of sentences. For each tag in the set of tags it also initialises a state which will then be tracked and used to calculate transition and emission changes in the state.
- The 'initTransitionMatrix' initialises the transition matrix frequency i.e. it counts the number of times a tag appears and adds it to the matrix array
- The 'train' method then proceeds to train the HMM model with a set of tagged sentences parsed from the main.py script. It takes two sentences at ones and splits the words into current word, current tag, next word, and next tag pairs. It then, for each tag, inserts the word and creates a transition matrix for those set of pairs where the transitional matrix becomes dependent on the current tag encoding and the next tag encoding. This is done iteratively for all tags in each sentence.
- The transitional matrix, (a conditional frequency distribution) now filled with integers, is then normalised on each row within the table iteratively to get conditional probability numbers for next state, given current state.
- It then uses the Viterbi Algorithm to find the optimum sequences between tag transitions. The algorithm initialises and uses recursion to predict the transition and emission matrices.
- The model is then evaluated using the 'evaluate' method after which it returns a percentage accuracy.

¹ Hidden Markov Model

Testing

- Using the floresta corpus took a fairly longer time to compute and evaluate.
- In testing, the following 3 corpora were used: Brown, Conell-2000 and Alpino
- Attached below is the results of the test



```
Activities Terminal
soo3@pc2-072-l:~/CS5012
File Edit View Search Terminal Help
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
Downloading Training Sentences from Corpus
Done!
Downloading Test Sentences from Corpus
Done!
Separating Words and Tags
Done!
Creating HMMs
Training HMM Model with POS Tags
Done!
Calculating Evaluation: TRAINING Sentences
Brown
97.86% accuracy
Cornell
98.51% accuracy
Alpino
93.72% accuracy
Floresta
Calculating Evaluation: TEST Sentences
Brown
91.74% accuracy
Cornell
91.10% accuracy
Alpino
Traceback (most recent call last):
```

- From the above results, the accuracy of brown and Alpino returned 91.74% and 91.10%
- Due to other deadlines and time constraints only, these tests were managed.

References

- https://github.com/spryor/Natural-Language-Processing/tree/master/Python/HMM_POS_Tagger
- https://en.wikipedia.org/wiki/Viterbi_algorithm
-